Introduction to Sharding in PostgreSQL

Presented by Dr. Smith

What is Sharding?

- A database architecture pattern that distributes data across multiple database instances or physical locations.
- Each shard contains a subset of the total data, reducing the load on any single server and improving performance.
- Many modern databases, including PostgreSQL, support sharding to achieve high scalability and performance.
- We will explore the benefits, strategies, and challenges of sharding in PostgreSQL.

Why is Sharding Important?

- **Scalability:** Facilitates handling of larger datasets and user loads by distributing them across multiple servers.
- **Performance:** Improves query response times by reducing the data volume each server needs to process.
- **Availability:** Increases resilience and uptime by ensuring no single point of failure (NOTE: Only when in used in with replication strategies).

When is Sharding Beneficial?

- **Large Datasets:** When datasets are too large for a single server to handle efficiently.
- **High Write/Read Throughput:** When the application demands high levels of read/write operations that a single server cannot provide.
- **Geographical Distribution:** When data needs to be located close to users to reduce latency.

When is Sharding Not Recommended?

- **Small Datasets:** Overhead may not justify the benefits.
- **Large number of Complex Transactions:** Cross-shard transactions can be complex and reduce performance.
- **Early Stage Projects:** Premature optimization can lead to unnecessary complexity.

Sharding Strategies

- **Hash-based Sharding:** Distributes data across shards based on a hash of the primary key.
- **Range-based Sharding:** Distributes data across shards based on a range of values in a specific column.
- **List-based Sharding:** Distributes data across shards based on a list of values in a specific column.
- **Composite Sharding:** Combines multiple strategies to distribute data across shards.

SQL Commands Involved in Sharding

- CREATE TABLE (with PARTITION BY)
- CREATE SERVER
- CREATE USER MAPPING
- CREATE FOREIGN DATA WRAPPER
- CREATE FOREIGN TABLE or IMPORT FOREIGN SCHEMA

CREATE TABLE (with PARTITION BY)

```sql
-- create master partitioned table
CREATE TABLE IF NOT EXISTS temperatures (
    reading_date date,
    city VARCHAR(100),
    temp DECIMAL
) PARTITION BY RANGE (reading_date);
```

CREATE SERVER

```
CREATE SERVER IF NOT EXISTS node2
  FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS (host 'postgres_node2', dbname 'shard2');
```

CREATE USER MAPPING

```
CREATE USER MAPPING IF NOT EXISTS FOR user1
  SERVER node2
  OPTIONS (user 'user2', password 'password2');
```

CREATE FOREIGN TABLE or IMPORT FOREIGN SCHEMA

```
IMPORT FOREIGN SCHEMA node2_schema LIMIT TO (temps_2023)
    FROM SERVER node2 INTO node1_schema;
```

NOTE: Create foreign table is similar to creating a regular table, but it references a table in a different server. Currently, you cannot create a foreign table with a primary key or unique constraint; therefore, IMPORT FOREIGN SCHEMA is used to import the schema of the foreign table into the local database (which can include primary keys and unique constraints).

Postgres Extensions also offer Sharding Capabilities

- **PostgresXL:** An open-source extension that provides sharding capabilities for PostgreSQL.
- **Citus:** A distributed database that extends PostgreSQL to enable sharding and replication.
- **pg_shard:** A sharding extension for PostgreSQL that allows you to distribute data across multiple servers.

The Challenges of Sharding

- **Complexity:** Sharding adds complexity to the database architecture, making it harder to manage and maintain.
- **Data Distribution:** Ensuring data is evenly distributed across shards can be challenging.
- **Cross-Shard Queries:** Queries that require data from multiple shards can be complex and slow.
- **Data Consistency:** Ensuring data consistency across shards can be challenging.
- **Shard Failure:** Handling shard failures and ensuring high availability can be complex.

Conclusion

- Sharding is a powerful strategy for scaling databases by distributing data across multiple servers.
- It's most effective for large, high-traffic applications requiring high availability and low latency.
- Careful consideration is required to determine if sharding is appropriate for your specific use case, given its complexity and overhead.

Next week, we will look at Cassandra, a distributed NoSQL database that uses sharding to achieve high scalability and performance (in a much less complex way than PostgreSQL!)

Q&A

- Feel free to ask any questions or share your thoughts on PostgreSQL sharding.